GAPFORMER: Fast Autoregressive Transformers meet RNNs for Personalized Adaptive Cruise Control

Noveen Sachdeva[1] Ziran Wang[2] Kyungtae Han[2] Rohit Gupta[2] Julian McAuley[1]

***Abstract*— Adaptive cruise control has been an important function in modern vehicles, and has proven to be helpful for assisted driving. The main challenges involve accurate gap prediction between the ego and preceding vehicles, as well as personalizing the driving behaviour for different kinds of drivers and/or cars. Correspondingly, in this paper, we make the following contributions: (1) we propose GAPFORMER which combines the Transformer and RNN architectures to better model and personalize driving behaviour; (2) make necessary modifications to the Transformer attention mechanism for scaling to long driving contexts in a resource-efficient manner; and (3) propose an architecture-agnostic model training regime, HORIZON which improves generalization by incorporating a time-horizon and makes the models more accurate and robust. Detailed experiments on both public and proprietary datasets demonstrate that GAPFORMER can be up to 50% more accurate when compared to other ACC baselines, demonstrating its efficacy and potential for real-world application.**

I. INTRODUCTION AND BACKGROUND

A. Introduction

The emergence of Adaptive Cruise Control (ACC) systems can be traced back to the last century, when automotive companies started equipping their vehicles with ACC in the late 90s [1]. The core concept of an ACC system is to automatically adjust the ego vehicle speed to maintain a safe gap from the preceding vehicle, with the help of the perceived information from on-board perception sensors (*e.g.*, radar, LIDAR, and/or camera). However, commercially available ACC systems only allow drivers to choose from three or four pre-determined car-following gap settings (*e.g.*, short, medium, long, and/or extra long), without considering each driver's preference in terms of driving and riding [2]–[4].

To this effect, in this paper, we aim to *personalize* the ACC functionality, where we want to dynamically infer the user's most preferable car-following gap based on a wide list of covariates such as their previous driving interactions, current weather, geography, *etc*. More specifically, we take a data-driven approach towards personalized ACC, where we build and learn a parametric model to estimate each driver's car-following preferences by learning from historical driving data (see Figure 1 for an overview).

Corresponding author: Noveen Sachdeva, nosachde@ucsd.edu

[1]CSE Department, University of California San Diego, La Jolla, CA 92122.

[2]InfoTech Labs, Toyota Motor North America R&D, Mountain View, CA 94043.

B. Personalized Adaptive Cruise Control (P-ACC)

For ACC systems, physics-based policies are the most dominant control policies at the current stage. Ordinary Differential Equation (ODE) algorithms (*e.g.*, Gipps model [5], Intelligent Driver Model (IDM) [6], and Newell's car-following model [7]) are proposed to allow the ego vehicle to follow the preceding vehicle's movement based on a certain set of parameters, such as the distance gap between two vehicles and speeds of both vehicles. Model Predictive Control (MPC) is another popular physics-based algorithm for ACC, which optimizes the predefined objectives (*e.g.*, safety, comfort, fuel efficiency, *etc.*) in a receding-horizon fashion [8]. However, both ODE and MPC policies require prior knowledge of the car-following system to design corresponding algorithms, deeming both policies to be highly generic and oftentimes difficult to personalize. Another limitation of ODE and MPC policies is their lack of expressivity — majority of the times, naturalistic human-driving does not follow their premeditated set of rules and contains exquisite nuances that cannot be captured by these algorithms.

More recently, learning-based algorithms have attracted more attention in the research domain of ACC systems. These methods generally perform better at driver trajectory modeling than physics-based policies, and they are also less restrictive and can be trained without any explicit assumptions about the car-following system. A major part of relevant studies adopt Inverse Reinforcement Learning (IRL) to learn the reward of the car-following demonstration trajectories, and then use controllers to implement the recovered reward. The works by Gao et al. [9] and Zhao et al. [10] both show that their IRL algorithms are capable of recovering the personalized car-following gap preference based on different vehicle speed values, where this gap-speed matrix can be used to design the downstream control logic for P-ACC systems. Another category of relevant studies directly look into the data and learn from the demonstration trajectories. For example, Wang et al. developed a Gaussian Process Regression algorithm for P-ACC, where both numerical and human-in-the-loop experiments verify the effectiveness of the proposed algorithm in terms of reducing the interference frequency by the driver [11]. Additionally, because the decision-making process of human drivers depends on sequential state inputs, Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) were also adopted to model the car-

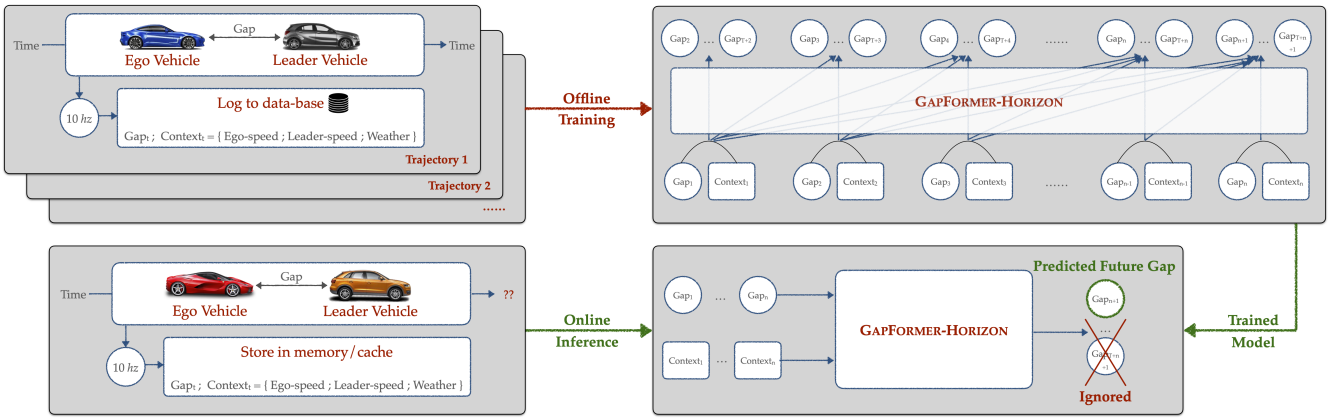


Fig. 1: System architecture of the proposed P-ACC framework. We train GapFormer-Horizon on a collection of logged human driving trajectories, and deploy it in real-world driving scenarios.

following behaviors [12], [13].

C. Sequence Modeling

The problem of naturalistic car-following trajectory modeling overlaps substantially with that of user journey modeling in the context of sequential recommender systems and personalization [14]–[17]. The main source of input for sequential recommendation is the chronological sequence of every user's interactions with a content delivery service like Netflix, YouTube, *etc.*; which the model uses as context to dynamically infer the user's interests for the next-item prediction. Historically, Markov-chain based methods like FPMC [18], Fossil [19], *etc.* operated on the first-order assumption that the next item for a user can be inferred by *just* looking at the previous item they consumed. Consequently, recent models like GRU4Rec [20], SVAE [15], *etc.* propose to use an RNN to embed the entire user consumption sequence.

More recently, the Transformer [21] architecture (originally proposed for NLP tasks such as large-scale language modeling), has seen immense success in a wide variety of general machine learning sequence-prediction tasks such as NLP [22], recommendation [14], [23], Speech [24], *etc.* However, their application to the field of driver trajectory modeling is still unexplored. These models fundamentally operate on the self-attention phenomenon [25] which suggests that every item can be viewed as a complex, dynamically-weighted function of the input sequence itself. Self-attention also inherits the property of being natively parallel (unlike inherently linear RNNs), thereby leveraging the parallel processing capabilities of recent hardware innovations like GPUs, TPUs, *etc.* However, attending to all previous tokens for every token is a quadratic process and hence takes $\mathcal{O}(n^2)$ memory; this prohibits transformer-based models from scaling to larger sequence lengths. Consequently, another line of research focuses on scaling up the transformer architecture by sparsifying the attention connections, thereby allowing models to incorporate larger context [26]–[28].

D. Contributions

Two prominent directions toward improving trajectory modeling include: (1) building more accurate and more robust driver trajectory models; and (2) since many of these models will be deployed on edge devices—improving the memory and run-time requirements of such models during inference. To this effect, *in this paper*, we:

- Propose GapFormer which combines the sequence modeling capabilities of a Transformer [21] and a Recurrent Neural Network (RNN) [29] for driver trajectory modeling. Our experiments demonstrate that such a combination leads to a better performance than using RNNs or Transformers individually.
- Modify the core self-attention architecture of a naïve Transformer block by only attending to a fixed number of previous tokens to ameliorate its $\mathcal{O}(n^2)$ memory complexity. This allows GapFormer to attend to longer contexts, leading to better down-stream performance while being significantly faster to train and infer.
- Propose Horizon, which is a novel training regime for any autoregressive architecture, and is especially beneficial for our safety-critical task of trajectory modeling, where the model should ideally plan for a distant future horizon rather than predicting the optimal conditions merely for the next time-step.

To validate our approach, we compare GapFormer with a variety of baselines and state-of-the-art models on both public and proprietary datasets. Experiments demonstrate that (1) GapFormer can be up to 50% more accurate compared to other ACC baselines; (2) GapFormer can be an order of a magnitude faster at inference compared to the vanilla Transformer; and (3) Horizon boosts performance

up to 15% across all architectures compared in this paper, deeming HORIZON a suitable choice for training autoregressive architectures.

II. METHODOLOGY

A. Problem Statement

Given a driver u along with his/her chronological sequence of gap between the ego and preceding vehicles $\mathcal{G}^u = (\mathcal{G}^u_1, \mathcal{G}^u_2, \cdots \mathcal{G}^u_{|\mathcal{G}^u|})$ where $\mathcal{G}^u_i \in \mathbb{R}^+ \; \forall i \in [1, |\mathcal{G}^u|]$; along with some environmental context sequence (*e.g.* weather, leader speed, *etc.*) $\mathcal{C}^u = (\mathcal{C}^u_1, \mathcal{C}^u_2, \cdots \mathcal{C}^u_{|\mathcal{G}^u|})$ where $\mathcal{C}^u_i \in \mathbb{R}^c \; \forall i \in [1, |\mathcal{G}^u|]$; predict the next, most optimal gap for the driver *i.e.* $\mathcal{G}^u_{|\mathcal{G}^u|+1} \in \mathbb{R}^+$.

An overview of our P-ACC framework is presented in Figure 1, where we demonstrate how GAPFORMER can be trained on the cloud and can be deployed for inference on edge devices. Specifically for offline training, a collection of driving trajectories are uploaded to a database on the cloud, and GAPFORMER can be trained on these collection of trajectories using a multi-GPU environment for accelerated training. Then, the trained model is deployed for online inference, where each vehicle's *current* trajectory takes a single forward pass through GAPFORMER, and the optimal future gap is predicted. This output will serve as the input of the downstream P-ACC control algorithm, which falls out of the scope of this paper.

B. GAPFORMER: *Fast Autoregressive Transformers meet RNNs for Personalized Driver Trajectory Modeling*

1) Embedding layer: Given the gap sequence $\mathcal{G}^u$ and the context sequence $\mathcal{C}^u$, we first transform both to have a fixed length n, which we tune as a hyper-parameter. If $|\mathcal{G}^u| > n$ we keep the most recent n actions, otherwise we pad both $\mathcal{G}^u, \mathcal{C}^u$ with the required number of padding elements (from the left). Given the padded gap and context sequences, we then concatenate them followed by an affine Multi-Layer Perceptron (MLP) transformation to obtain an intermediate representation:

$$\hat{\mathcal{S}}^u = \left(W_e^T \cdot (\mathcal{G}^u_i \;||\; \mathcal{C}^u_i) + b_e \mid \forall i \in [1, |\mathcal{G}^u|]\right), \quad (1)$$

where $W_e \in \mathbb{R}^{(c+1)\times d}$ and $b_e \in \mathbb{R}$ represent the parameters of the embedding transformation, and '||' represents the concatenation operation. Consequently, we also inject a learnable position embedding, which is meant to inherently learn the dynamics of different positions through an embedding matrix $P_e \in \mathbb{R}^{n\times d}$ to get the final input embedding $\mathcal{S}^u = \hat{\mathcal{S}}^u + P_e^T$.

2) GRU: Given the embedded input sequence $\mathcal{S}^u$, we employ a Gated Recurrent Unit (GRU) [29] to embed it in a lower dimensional latent space. More formally, the GRU maintains a hidden-state vector $h_t \in \mathbb{R}^d$ for each time-step in the sequence and updates it:

$$h_{t+1} = \text{GRU}(h_t, \mathcal{S}^u_t),$$

where GRU represents the standard set of update-gate and reset-gate equations. We finally obtain a fixed-size representation of the *entire* sequence by extracting only the last hidden-state as processed by the GRU *i.e.* $\mathcal{E}_{\text{GRU}} \coloneqq h_{|\mathcal{G}^u|}$.

3) Transformer: We re-use the same embedded input sequence $\mathcal{S}^u$ as input for the Transformer component as well. We first perform multi-head attention [21] on $\mathcal{S}^u$ by splitting the input into multiple segments at each time-step and applying attention to each of them individually. After computing the attention function on each segment, we concatenate the resulting vectors again to get the final attentive intermediate representation $\mathcal{H}^a_t$ for each time-step. More formally, given h heads:

$$\begin{aligned} \mathcal{H}^a_t &= [\ \text{head}_0 \;||\; \text{head}_1 \;||\; \ldots \;||\; \text{head}_h\ ], \\ \mathcal{S}^u_t &= \left[\ \mathcal{S}^u_{t,0} \;||\; \mathcal{S}^u_{t,1} \;||\; \ldots \;||\; \mathcal{S}^u_{t,h}\ \right], \\ \text{head}_i &= \text{Attention}(\mathcal{S}^u_{t,i}, \mathcal{S}^u_{t,i}, \mathcal{S}^u_{t,i}), \end{aligned} \quad (2)$$

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{Q \cdot K^T}{\sqrt{d/h}}\right) \cdot V \quad (3)$$

Note that we do not use any further affine transformations on the attention inputs $\mathcal{S}^u_{t,i}$ in Equation (2) before sending it to the Attention layer, as we otherwise noted large amounts of model overfitting in our experiments. We also point out that we explicitly mask the attention matrix in Equation (3) with a lower triangular matrix to not include connections for *future* time-steps and maintain causality. Note however, that the overall process in Equation (3) is still linear. To this end, Transformer adds a series of non-linear, residual, feed-forward network layers to increase model capacity, as follows:

$$\begin{aligned} \mathcal{H}_t &= \mathcal{H}^a_t + \mathcal{H}^n_t, \\ \mathcal{H}^n_t &= \text{Conv}_2(\text{ReLU}(\text{Conv}_1(\mathcal{H}^a_t))), \end{aligned} \quad (4)$$

where Conv_1 and Conv_2 are parametrized by different bi-linear parameters $W_1, W_2 \in \mathbb{R}^{d\times d}$. We finally obtain a fixed-size representation of the *entire* sequence by extracting only the last hidden-state as processed by the Transformer *i.e.* $\mathcal{E}_{\text{Transformer}} \coloneqq \mathcal{H}_{|\mathcal{G}^u|}$.

4) Linear self-attention: Even though the Transformer architecture has seen success in a wide variety of applications, it is still inherently quadratic and requires $\mathcal{O}(n^2)$ memory as demonstrated by Equation (3) where there is a Softmax call over an $n \times n$ attention matrix. Inspired by recent literature [26], [27], which suggests that only a relatively short window of previous token connections are needed for generalization in the Transformer architecture, we follow a sliding-window based attention restriction mechanism that only attends to χ previous elements for every element in the sequence. We provide a visualization of our restriction mechanism in Figure 2. This provides us with major benefits like: (1) better downstream generalization and performance of GAPFORMER; and (2) linear memory requirements during

training and inference allowing GAPFORMER to scale to long driving trajectories (see Figure 5 for further details).

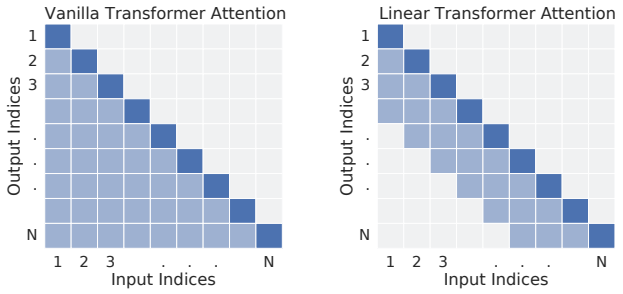


Fig. 2: Comparison of vanilla transformer attention *vs.* GAPFORMER's attention for an input sequence of size N, $\chi = 3$.

5) Fusion layer: Even though $\mathcal{E}_{\text{Transformer}}$ can be computed in a compute-efficient manner, it still lacks very distant context by design, and trades-off context length with compute-efficiency. To this end, we propose to combine $\mathcal{E}_{\text{Transformer}}$ with $\mathcal{E}_{\text{GRU}}$, which does have access to the *entire* driving trajectory. We argue that this allows GAPFORMER to enjoy the best of both worlds — have a very sophisticated modeling of the shorter driving context ($\mathcal{E}_{\text{Transformer}}$), and an easy-to-scale representation of the distant context ($\mathcal{E}_{\text{GRU}}$); thereby allowing the final context representation to be highly accurate and yet easy-to-scale. More formally, to obtain the final context representation, we dynamically infer the importance of the GRU and Transformer architectures' embeddings, by learning two scalar parameters $\alpha, \beta \in \mathbb{R}$ and performing a weighted average:

$$\begin{aligned} \mathcal{E} &= (\alpha' \cdot \mathcal{E}_{\text{GRU}}) + (\beta' \cdot \mathcal{E}_{\text{Transformer}}), \\ [\alpha', \beta'] &= \text{Softmax}([\alpha, \beta]) \end{aligned} \tag{5}$$

Having represented $\mathcal{S}^u$ in a small latent-space by $\mathcal{E} \in \mathbb{R}^d$, we now perform a series of non-linear affine transformations to predict the desired gap at the next time-step:

$$\hat{\mathcal{G}}^u_{|\mathcal{G}^u|+1} = \mathcal{G}^u_{|\mathcal{G}^u|} + \Delta u, \tag{6}$$

$$\Delta u = \left(F_2^T \cdot \text{ReLU}(F_1^T \cdot \mathcal{E} + b_1)\right) + b_2, \tag{7}$$

where $F_1 \in \mathbb{R}^{d \times d}$, $F_2 \in \mathbb{R}^{d \times 1}$, and $b_1, b_2 \in \mathbb{R}$ represent the parameters for decoding $\mathcal{E}$ to predict the desired gap. We also want to point out that we predict the *change in gap* from the previous time-step rather than the exact gap itself (Equation (7)), as doing so eliminates redundant learning and directly focuses on our driver trajectory modeling task.

6) Optimization problem: Having discussed the details about GAPFORMER's architecture, we now discuss how its parameters are optimized. To simplify notation let $\phi_\theta : (\mathcal{G}^u, \mathcal{C}^u) \rightarrow \mathbb{R}$ represent GAPFORMER. We optimize θ to minimize the point-wise MSE between the predicted and actual gaps:

$$\arg\min_\theta \sum_{u \in \mathcal{U}} \sum_{t=1}^{|\mathcal{G}^u|} \left(\phi_\theta(\mathcal{G}^u_{1:t}, \mathcal{C}^u_{1:t}) - \mathcal{G}^u_{t+1}\right)^2 + \lambda \cdot ||\theta||_2^2 \tag{8}$$

Where $\mathcal{U}$ represents the set of all drivers in our dataset, $\mathcal{G}^u_{1:t}$, $\mathcal{C}^u_{1:t}$ represent the gap and context trajectories for driver u up to the t^{th} timestep respectively, and λ is a regularization constant. We will optimize Equation (8) through standard mini-batch SGD.

7) Personalization: As we can see, there is no explicit driver representation in GAPFORMER. Previous work in the personalization literature either suggests the usage of (1) *explicit* user embeddings $\mathcal{U} \in \mathbb{R}^{n \times d}$, where we estimate a fixed $d-$dimensional representation for each of the n users in the dataset [30], [31]; or (2) *implicit* user modeling where we represent the user directly by the set of items they consume [14], [15], [23]. GAPFORMER follows the implicit user modeling strategy as the driver is strictly represented by their historical driving trajectory, and we aim to mine subtle driver traits directly from sequential information.

C. HORIZON: A Training Regime for Autoregressive Models

The core idea in HORIZON is to train the autoregressive model to predict the distance gaps for a *fixed* horizon of $\kappa-$steps in the future, rather than just the next timestep. The motivation for HORIZON is also relatively straightforward — we are forcing the model to learn the long-term trajectory dynamics and driver traits, rather than greedy next-gap prediction. An added benefit of HORIZON is that it will provide more gradient for better generalization, while being more stable and faster to converge. More formally, we change: (1) the decoder architecture in Equation (7), to predict for $\kappa-$steps in the future rather than 1, by changing the dimensionality of $F_2 \in \mathbb{R}^{d \times 1} \rightarrow F_2 \in \mathbb{R}^{d \times \kappa}$; and (2) the optimization problem in Equation (8) to the following:

$$\arg\min_\theta \sum_{u \in \mathcal{U}} \sum_{t=1}^{|\mathcal{G}^u|-\kappa} \sum_{i=1}^{\kappa} \left(\phi_{\theta,i}(\mathcal{G}^u_{1:t}, \mathcal{C}^u_{1:t}) - \mathcal{G}^u_{t+i}\right)^2 + \lambda \cdot ||\theta||_2^2 \tag{9}$$

Where $\phi_{\theta,i}$ represents the i^{th} output of ϕ. Note that HORIZON is generic and can be applied to any autoregressive model as long as it predicts future events through some set of learnable parameters.

III. EXPERIMENTS

A. Data

To evaluate the performance and robustness of GAPFORMER and HORIZON, we experiment on a public real-world, and a proprietary synthetic dataset:

- *Open-ACC [32]:* It is a popular car-following dataset originally intended to better understand the ACC characteristics of modern vehicles. Instead of the regular ego and preceding vehicle setup, Open-ACC collects data

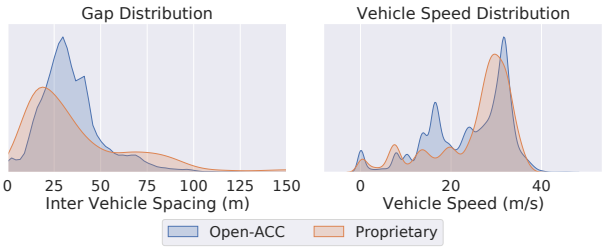


Fig. 3: Gap and vehicle speed distributions of both datasets.

where a platoon of vehicles are following each other. To convert the vehicle-platoon setup in Open-ACC to the ego and preceding vehicle setup we need for our experiments, we consider all vehicle trajectories in the platoon independently, giving us $n-1$ trajectories for each platoon trajectory of n vehicles.

- *Proprietary:* We also synthesize realistic data from an in-house, human-in-the-loop game engine built in previous work [33] over the Unity [34] platform. More specifically, we ask five different drivers to generate car-following data by either driving a midsize sedan *or* an SUV under varying weather conditions. We simulate clear, cloudy, foggy, and storm conditions as settings in the Unity game.

The driving trajectories for both datasets are sampled at $10Hz$ and characteristics like the vehicle type, current speed, lead-vehicle speed, distance gap, and time-gap are logged. Note that the weather is also available for the proprietary dataset. From Figure 3, we observe that both datasets follow very similar gap and speed distributions, with the major difference being only in the amount of total data and average size of the driving trajectory as compared in Table I.

We preprocess the data to only include those contextual features which are not directly dependent on the logging policy *i.e.* the lead vehicle speed, current weather, and vehicle type, and *remove* features like ego-vehicle speed, acceleration, *etc.* This is because these features share a *causal* link by the gap predictions our models make and will otherwise cause data-leakage when making predictions for our models. To the best of our knowledge, this seems like a limiting factor for any auto-regressive model, and incorporating these crucial, model output-dependent features seems to be an important avenue for future work. We then split both datasets into $80/10/10\%$ train/test/validation portions, respectively. More specifically, to retain the sequential nature of our data, we split *each* driving trajectory to have the first 80% of sequence to be in the train-set, next 10% to be in the validation-set, and the last 10% in the test-set. We also normalize all our features to follow a normal distribution since they can be arbitrarily large, but un-normalize the models' predicted gaps while reporting results.

TABLE I: Brief statistics of the datasets used in this paper.

Dataset	*Trajectories* Num.	Avg / Median length	Sampling Freq.	Simulated
Open-ACC	46	8.6k / 5.5k	10 Hz	×
Proprietary	20	2.4k / 2.7k	10 Hz	Yes

B. Evaluation Metric

To compare the performance of different models for our task of driver trajectory modeling, we employ an RMSE-based prediction horizon evaluation scheme, commonly seen in the P-ACC literature [35], [36]. More specifically, given a gap prediction function $\phi : (\mathcal{G}^u, \mathcal{C}^u) \to \mathbb{R}$, we define our metric of interest $\text{RMSE}_{\text{mean}}$ as:

$$\text{RMSE}_{\text{mean}} = \frac{1}{\tau} \cdot \sum_{t=1}^{\tau} \text{RMSE@}k, \tag{10}$$

$$\text{RMSE@}k = \sqrt{\sum_{u \in \mathcal{U}} \left(\phi_t(\mathcal{G}^u, \mathcal{C}^u) - \mathcal{G}^u_{|\mathcal{G}^u|+t} \right)^2},$$

where τ represents a fixed prediction horizon we want to evaluate our model on (we fix τ to be $100 \simeq 10$s for our experiments), $\mathcal{U}$ represents the set of all drivers in our test-set, and with a slight abuse of notation, $\phi_t(\mathcal{G}^u, \mathcal{C}^u)$ represents the prediction made by ϕ for the t^{th} time-step in the future.

C. Baselines

We compare our models with a wide set of baselines and competitor state-of-the-art methods as discussed below:

- *Copy:* This is the most practical baseline which maintains the *same gap* between the ego and preceding vehicles, irrespective of any context.
- *Linear:* This baseline is a smarter heuristic compared to the Copy-baseline, which continues making a step in the same direction, by the same amount, as the driver did before the current time-step. More formally, we predict the gap $\hat{\mathcal{G}}^u_{|\mathcal{G}^u|+t} = \mathcal{G}^u_{|\mathcal{G}^u|} + t \cdot (\mathcal{G}^u_{|\mathcal{G}^u|} - \mathcal{G}^u_{|\mathcal{G}^u|-1})$.
- *MLP:* Recent work proposed using a 3-layer MLP architecture for lane-change prediction [37]. We adapt their model to our task of gap-prediction by optimizing their architecture with the L2-regression loss mentioned in Equation (8). We note that this model is equivalent to the Markovian assumption *i.e.* the gap for a future time-step can be predicted by looking only at the *previous* time-step.
- *RNN:* Recent work in driver trajectory modeling [35] suggests using the Long Short-Term Memory (LSTM) architecture to predict the optimal acceleration for a driver. For the sake of comparison, we adapt their architecture to predict the gap rather than the acceleration.

TABLE II: $\text{RMSE}_{\text{mean}}$ values for all methods on both datasets. The best method for each dataset is **highlighted**. Methods ending with * are proposed in this paper.

	Baselines		*Non-sequential*		*Sequential*					
Dataset	**Copy**	**Linear**	**MLP**	**MLP** HORIZON	**RNN**	**Trans-former***	**GAP-FORMER***	**RNN** HORIZON	**Transformer** HORIZON *	**GAPFORMER** HORIZON *
Open-ACC	5.9707	4.4492	5.0851	4.7169	3.2906	3.5557	3.4146	3.1067	3.0031	**2.927**
Proprietary	4.9821	6.9529	4.4467	4.2437	4.3217	4.1989	4.1522	4.2167	4.1154	**4.077**

This architecture has been shown to work extremely well for driver trajectory modeling, represents the current state-of-the-art.

- *Transformer:* We also include the architecture where we solely use the Transformer component of GAPFORMER as a competitor. Note that in this model we still use the linear self-attention mechanism proposed in Section II-B.4, and not the vanilla Transformer's quadratic self-attention.

In addition to the aforementioned models, to demonstrate the efficacy of HORIZON as an architecture-agnostic training regime for autoregressive models, we also compare GAPFORMER with MLP-HORIZON, RNN-HORIZON, and Transformer-HORIZON wherein we train the corresponding architectures with the HORIZON loss mentioned in Equation (9).

D. Experiments

For fair comparison across algorithms, we perform an extensive grid-search over various hyper-parameters like latent size d, L2-regularization constant λ, SGD learning rate, HORIZON's κ, attention's sliding window length χ, and the maximum driving trajectory length. We now discuss our main research questions.

How do different models compare with each other?
In Table II we compare the performance of GAPFORMER with the different models discussed in Section III-C on both the Open-ACC and Proprietary datasets. We first note that both non-sequential and sequential models improve over the popular baselines, and that sequential methods outperform non-sequential methods by a significant margin. Next, we observe that GAPFORMER-HORIZON outperforms all other methods across datasets on the $\text{RMSE}_{\text{mean}}$ metric. This goes to show that combining the short-term modeling of the Transformer, along with the long-term modeling of the RNN (Section II-B.5) helps GAPFORMER get the best of both worlds, thereby being more accurate than both architectures individually. Finally, we also note that HORIZON significantly improves model performance across architectures, and shows consistent gains for the MLP, RNN, Transformer, and GAPFORMER architectures on both datasets.

Across how distant of a horizon can models generalize well?
To better understand the variability and generalization patterns of different models, instead of comparing them solely on $\text{RMSE}_{\text{mean}}$, we stratify the model evaluation for different numbers of time-steps in the future horizon, and compare the respective RMSE@k values as plotted in Figure 4. There are a few prominent observations to make. Firstly, the Linear baseline starts off as a good heuristic but as we stretch our prediction horizon, it becomes increasingly worse. Next, we notice that $\text{RMSE}_{\text{mean}}$ corresponds quite well with overall model performance across different horizons, which can be evidenced by the performance lines for different models staying in agreement with each other for a majority portion of the prediction horizon. Finally, it becomes evident that HORIZON significantly helps the generalization of all the model architectures we consider in this paper. To be more specific, the models and their HORIZON versions are initially similar but the difference in generalization increases significantly as we predict for a distant horizon.

How does GAPFORMER*'s sparse attention affect performance and inference?*
To compare the effect of the sparse attention mechanism as visualized in Figure 2, we plot the performance ($\text{RMSE}_{\text{mean}}$), memory footprint, and inference time of the linear and vanilla Transformers *vs.* the driving trajectory sequence length in Figure 5. We note that as we increase the sequence length (1) the performance of the linear transformer improves due to more context, whereas the vanilla transformer deteriorates (perhaps because of overfitting); and (2) the GPU memory requirements as well as the inference time increases drastically for the vanilla transformer making the architecture impractical for deployment on the modest amount of computation available even for a modern vehicle.

How do different models respond to perturbed inputs?
We conduct an experiment where we test the three best performing models *i.e.* RNN-HORIZON, Transformer-HORIZON, and GAPFORMER-HORIZON under different kinds of input perturbations on the Open-ACC dataset. More specifically, we first change the driving trajectory context of a randomly sampled driving trajectory to be a different vehicle and see how the trained models' predictions change. As we can see from the models' predictions visualized in Figure 6, the models do learn to personalize their predictions for different kinds of vehicles. For example, the models unanimously predict a shorter gap for a Mini-Cooper (subcompact vehicle) *vs.* a larger gap for the Tesla Model-X (SUV). We repeat the same procedure but now perturb the

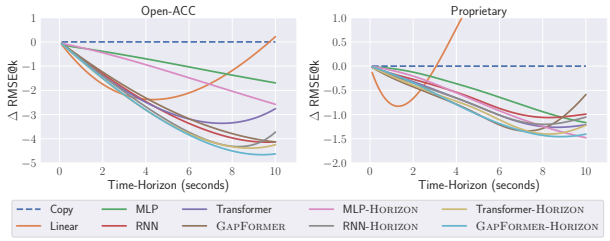
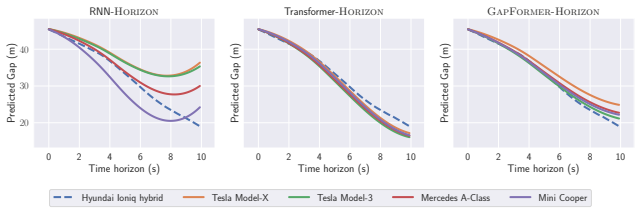


Fig. 4: Improvement in RMSE@k over the Copy baseline for various methods stratified over the prediction horizon.

Fig. 5: Comparison of $\text{RMSE}_{\text{mean}}$, GPU usage, and inference time *vs.* driving trajectory sequence length (log-scale) for linear and vanilla Transformers over the Open-ACC dataset.



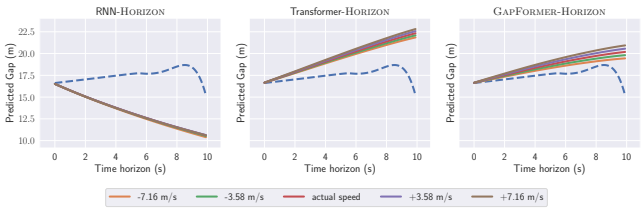


Fig. 6: Model predictions when the inputs are perturbed to *(left)* be different vehicles, or *(right)* have different leader speeds.

context to have a different lead vehicle speed. Similarly, we observe that the models again try to collectively predict a larger gap when the leader is at a higher speed *vs.* a smaller gap when at a lower speed. This phenomenon seems to be captured most prominently by GapFormer, again demonstrating its effectiveness over other state-of-the-art models.

IV. Conclusion

In this paper, we proposed GapFormer which efficiently combines the Transformer and RNN architectures for our task of personalized driver trajectory modeling. We also propose Horizon, which is a novel training regime for auto-regressive models and we empirically demonstrate its effectiveness for a variety of model architectures. Experiments on two public and proprietary datasets demonstrate that GapFormer can be up to 50% better than other ACC baselines while also being an order of magnitude faster at inference compared to the vanilla Transformer. Additionally, we note that Horizon improves generalization across architectures and can result in up to 15% performance gains by itself.

V. Future Work

For future work, we aim to further explore how to include policy-dependent contextual features like ego-vehicle speed, acceleration, *etc.* while predicting the future gap. This is a challenging problem as these features share a direct *causal* link with the autoregressive model's prediction outcome. Other important avenues we would like to explore in the future include adding more robust and expressive features like live driving feed, GPS, time-of-day, *etc.*

Acknowledgment

The contents of this paper only reflect the views of the authors, who are responsible for the facts and the accuracy of the data presented herein. The contents do not necessarily reflect the official views of Toyota Motor North America.

References

[1] G. Marsden, M. McDonald, and M. Brackstone, "Towards an understanding of adaptive cruise control," *Transportation Research Part C: Emerging Technologies*, vol. 9, no. 1, pp. 33–51, 2001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0968090X0000022X

[2] Toyota. (2021) Toyota Safety Sense: The Standard for Safety. [Online]. Available: https://www.toyota.com/safety-sense/

[3] Ford. (2021) Adaptive Cruise Control. [Online]. Available: https://www.ford.com/technology/driver-assist-technology/adaptive-cruise-control/

[4] Volkswagen. (2021) Adaptive Cruise Control. [Online]. Available: https://www.volkswagen-newsroom.com/en/adaptive-cruise-control-acc-3664

[5] P. Gipps, "A behavioural car-following model for computer simulation," *Transportation Research Part B: Methodological*, vol. 15, no. 2, pp. 105–111, 1981. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0191261581900370

[6] A. Kesting, M. Treiber, and D. Helbing, "Enhanced intelligent driver model to access the impact of driving strategies on traffic capacity," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 368, no. 1928, pp. 4585–4605, 2010.

[7] G. F. Newell, "A simplified car-following theory: a lower order model," *Transportation Research Part B: Methodological*, vol. 36, no. 3, pp. 195–205, 2002.

[8] S. Li, K. Li, R. Rajamani, and J. Wang, "Model predictive multi-objective vehicular adaptive cruise control," *IEEE Transactions on control systems technology*, vol. 19, no. 3, pp. 556–566, 2010.

[9] H. Gao, G. Shi, G. Xie, and B. Cheng, "Car-following method based on inverse reinforcement learning for autonomous vehicle decision-making," *International Journal of Advanced Robotic Systems*, vol. 15, no. 6, p. 1729881418817162, 2018.

[10] Z. Zhao, Z. Wang, K. Han, P. Tiwari, G. Wu, and M. Barth, "Personalized car following for autonomous driving with inverse reinforcement learning," in *Proceedings 2022 IEEE International Conference on Robotics and Automation*, 2022.

[11] Y. Wang, Z. Wang, K. Han, P. Tiwari, and D. B. Work, "Personalized adaptive cruise control via gaussian process regression," in *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, 2021, pp. 1496–1502.

[12] L. Chong, M. M. Abbas, and A. Medina, "Simulation of driver behavior with agent-based back-propagation neural network," *Transportation Research Record*, vol. 2249, no. 1, pp. 44–51, 2011.

[13] X. Huang, J. Sun, and J. Sun, "A car-following model considering asymmetric driving behavior based on long short-term memory neural networks," *Transportation research part C: emerging technologies*, vol. 95, pp. 346–362, 2018.

[14] W.-C. Kang and J. McAuley, "Self-attentive sequential recommendation," in *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2018, pp. 197–206.

[15] N. Sachdeva, G. Manco, E. Ritacco, and V. Pudi, "Sequential variational autoencoders for collaborative filtering," *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, 2019.

[16] N. Sachdeva, K. Gupta, and V. Pudi, "Attentive neural architecture incorporating song features for music recommendation," *Proceedings of the 12th ACM Conference on Recommender Systems*, 2018.

[17] N. Sachdeva, C.-J. Wu, and J. McAuley, "On sampling collaborative filtering datasets," in *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, ser. WSDM '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 842–850.

[18] S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme, "Factorizing personalized markov chains for next-basket recommendation," in *Proceedings of the 19th international conference on World wide web*, 2010, pp. 811–820.

[19] R. He and J. McAuley, "Fusing similarity models with markov chains for sparse sequential recommendation," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 2016, pp. 191–200.

[20] D. Jannach and M. Ludewig, "When recurrent neural networks meet the neighborhood for session-based recommendation," in *Proceedings of the Eleventh ACM Conference on Recommender Systems*, 2017, pp. 306–310.

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[22] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. [Online]. Available: https://aclanthology.org/N19-1423

[23] F. Sun, J. Liu, J. Wu, C. Pei, X. Lin, W. Ou, and P. Jiang, "Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer," in *Proceedings of the 28th ACM international conference on information and knowledge management*, 2019, pp. 1441–1450.

[24] L. Dong, S. Xu, and B. Xu, "Speech-transformer: A no-recurrence sequence-to-sequence model for speech recognition," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018, pp. 5884–5888.

[25] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *CoRR*, vol. abs/1409.0473, 2015.

[26] M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang *et al.*, "Big bird: Transformers for longer sequences," *Advances in Neural Information Processing Systems*, vol. 33, pp. 17 283–17 297, 2020.

[27] S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma, "Linformer: Self-attention with linear complexity," *arXiv preprint arXiv:2006.04768*, 2020.

[28] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret, "Transformers are rnns: Fast autoregressive transformers with linear attention," in *International Conference on Machine Learning*. PMLR, 2020, pp. 5156–5165.

[29] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[30] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[31] J. Tang and K. Wang, "Personalized top-n sequential recommendation via convolutional sequence embedding," in *Proceedings of the eleventh ACM international conference on web search and data mining*, 2018, pp. 565–573.

[32] M. Makridis, K. Mattas, A. Anesiadou, and B. Ciuffo, "Openacc. an open database of car-following experiments to study the properties of commercial acc systems," *Transportation Research Part C: Emerging Technologies*, vol. 125, p. 103047, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0968090X21000772

[33] Z. Wang, K. Han, and P. Tiwari, "Digital twin simulation of connected and automated vehicles with the unity game engine," in *2021 IEEE 1st International Conference on Digital Twins and Parallel Intelligence (DTPI)*, 2021, pp. 1–4.

[34] Unity, "Unity for all." [Online]. Available: https://unity.com/

[35] I. Jones and K. Han, "Probabilistic modeling of vehicle acceleration and state propagation with long short-term memory neural networks," in *2019 IEEE Intelligent Vehicles Symposium (IV)*, 2019, pp. 2236–2242.

[36] J. Su, P. A. Beling, R. Guo, and K. Han, "Graph convolution networks for probabilistic modeling of driving acceleration," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2020, pp. 1–8.

[37] Z. Shou, Z. Wang, K. Han, Y. Liu, P. Tiwari, and X. Di, "Long-term prediction of lane change maneuver through a multilayer perceptron," in *2020 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2020, pp. 246–252.